UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/675,265	09/29/2003	Kevin Brown	SVL920030045US1	4880

47069 7590 06/17/2009
KONRAD RAYNES & VICTOR, LLP
ATTN: IBM54
315 SOUTH BEVERLY DRIVE, SUITE 210
BEVERLY HILLS, CA 90212

EXAMINER	
TIMBLIN, ROBERT M	

ART UNIT	PAPER NUMBER
2167	

NOTIFICATION DATE	DELIVERY MODE
06/17/2009	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

krvuspto@ipmatters.com

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS
AND INTERFERENCES

Ex parte KEVIN BROWN and RAGHUPATHI KESHAVA MURTHY

Appeal 2008-004827
Application 10/675,265
Technology Center 2100

Decided:[1] June 15, 2009

Before JOSEPH L. DIXON, LANCE LEONARD BARRY, and CAROLYN D. THOMAS, *Administrative Patent Judges.*

DIXON, *Administrative Patent Judge.*

DECISION ON APPEAL

---

[1] The two-month time period for filing an appeal or commencing a civil action, as recited in 37 C.F.R. § 1.304, begins to run from the decided date shown on this page of the decision. The time period does not run from the Mail Date (paper delivery) or Notification Date (electronic delivery).

I. STATEMENT OF THE CASE

A Patent Examiner rejected claims 1-39. The Appellants appeal therefrom under 35 U.S.C. § 134(a). We have jurisdiction under 35 U.S.C. § 6(b).

We REVERSE.

Invention

The invention at issue on appeal relates to a method, system, and computer program of database system for processing predicates in an iterator function, where the iterator function included in a query statement is invoked (Spec. 4).

Illustrative Claim

Claim 1, which further illustrates the invention, is as follows:

> 1. A method for processing predicates in an iterator function, comprising:
>
> under control of an iterator function processor that executes as part of a data store engine, when an iterator function included in a statement is invoked,
>
> obtaining one or more predicates included in the statement; applying the one or more predicates to a row of data; if applying the one or more predicates results in a match, returning the row of data; and
>
> if applying the one or more predicates does not result in a match, searching for another row of data for which application of the one or more predicates results [sic] in a match.

Reference

The Examiner relies on the following reference as evidence:

Andrei	US 6,801,905 B2	Oct. 5, 2004

Rejection

The Examiner makes the following rejections.

Claims 1-39 stand rejected under 35 U.S.C. § 102(e) as anticipated by Andrei.

II. ISSUES

Has the Examiner set forth a sufficient initial showing of anticipation of the claimed invention? The issue regarding the independent claims turns on whether Appellants have shown the Examiner erred by failing to identify a teaching of when an iterator function included in a statement is invoked and then at least one predicate is applied to the row of data.

III. PRINCIPLES OF LAW

Prima Facie Case of Unpatentability

The allocation of burdens requires that the USPTO produce the factual basis for its rejection of an application under 35 U.S.C. §§ 102 and 103. *In re Piasecki*, 745 F.2d 1468, 1472 (Fed. Cir. 1984) (citing *In re Warner*, 379 F.2d 1011, 1016 (CCPA 1967)). The one who bears the initial burden of presenting a prima facie case of unpatentability is the Examiner. *In re Oetiker*, 977 F.2d 1443, 1445 (Fed. Cir. 1992). Appellants have the opportunity on appeal to the Board to demonstrate error in the Examiner's position. *See In re Kahn*, 441 F.3d 977, 985-86 (Fed. Cir. 2006).

Scope of Claim

The claim construction analysis begins with the words of the claim. *See Vitronics Corp. v. Conceptronic, Inc.*, 90 F.3d 1576, 1582 (Fed. Cir. 1996). Absent an express intent to impart a novel meaning to a claim term, the words take on the ordinary and customary meanings attributed to them by those of ordinary skill in the art. *Brookhill-Wilk 1, LLC. v. Intuitive Surgical, Inc.*, 334 F.3d 1294, 1298 (Fed. Cir. 2003) (citation omitted).

Anticipation

In rejecting claims under 35 U.S.C. § 102, "[a] single prior art reference that discloses, either expressly or inherently, each limitation of a claim invalidates that claim by anticipation." *Perricone v. Medicis Pharm. Corp.,* 432 F.3d 1368, 1375 (Fed. Cir. 2005) (citation omitted).

"[A]nticipation of a claim under § 102 can be found only if the prior art reference discloses every element of the claim . . . ." *In re King*, 801 F.2d 1324, 1326 (Fed. Cir. 1986) (citing *Lindemann Maschinenfabrik GMBH v. American Hoist & Derrick Co.,* 730 F.2d 1452, 1458 (Fed. Cir. 1984)). "[A]bsence from the reference of any claimed element negates anticipation." *Kloster Speedsteel AB v. Crucible, Inc.,* 793 F.2d 1565, 1571 (Fed. Cir. 1986), *overruled on other grounds by Knorr-Bremse Systeme Fuer Nutzfahrzeuge GmbH v. Dana Corp.*, 383 F.3d 1337 (Fed.Cir. 2004).

IV. FINDINGS OF FACT

In our analysis *infra*, we will rely on the following Findings of Fact (FF) that are supported by a preponderance of the evidence.

Andrei

1. Andrei discloses the general process for executing a query statement as follows:

> In operation, the SQL[Structured Query Language] statements received from the client(s) 310 (via network 320) are processed by engine 360 of the database server system 340. Engine 360 itself comprises parser 361, normalizer 363, compiler 365, execution unit 369, and access methods 370. Specifically, *the SQL statements are passed to the parser 361 which converts the statements into a query tree--a binary tree data structure which represents the components of the query in a format selected for the convenience of the system.* In this regard, the parser 361 employs conventional parsing methodology (e.g., recursive descent parsing).
>
> The query tree is normalized by the normalizer 363. Normalization includes, for example, the elimination of redundant data. Additionally, the normalizer 363 performs error checking, such as confirming that table names and column names which appear in the query are valid (e.g., are available and belong together). Finally, the normalizer can also look-up any referential integrity constraints which exist and add those to the query.
>
> After normalization, the query tree is passed to the compiler 365, which includes an optimizer 366 and a code generator 367. The optimizer is responsible for optimizing the query tree. The optimizer performs a cost-based analysis for formulating a query execution plan. The optimizer will, for instance, select the join order of tables (e.g., when working with more than one table); it will select relevant indexes (e.g., when indexes are available). *The optimizer, therefore, performs an analysis of the query and picks the best execution plan, which in turn results in particular ones of the access methods being invoked during query execution.*

> The code generator, on the other hand, converts the query tree into a set of instructions suitable for satisfying the query. These instructions are passed to the execution unit 369. Operating under the control of these instructions, *the execution unit 369 generates calls into lower-level routines, such as the access methods 370, for retrieving relevant information (e.g., row 355) from the database table 350.* After the plan has been executed by the execution unit, the server returns a query result or answer table back to the client(s).

(Fig. 3, col. 11, ll. 4-41) (Emphases added).

2. Andrei discloses in Figs. 8A and 8B a process and steps of eager enforcement (Col. 7, ll. 41-44) from receiving a query statement (step 801) to generating an optimized query execution plan (step 810) (Figs. 8A and 8B).

3. Andrei discloses a method and a database system for eager and opportunistic property enforcement (Col. 4, l. 57-col. 5, l. 8) enabling improved query optimization to form a query execution plan (hereinafter QEP), *i.e.*, a demand-drive tuple stream iterator tree that is a data structure to be interpreted by a database engine (Col. 6, ll. 23-41, Abstract, Col. 6, ll. 59-64).

4. Andrei further discloses the illustrating components of the optimization system in Fig. 7 which consist of the search engine 740, the logical properties module 720 including logical operator trees, and *the physical operator module 730 containing the hierarchy of physical operators and optimization time descriptions of physical operators that are used for optimization of the Volcano runtime iterators*. Those optimization time physical operators contain a description of the physical properties, such as the ordering, the partitioning, etc. The search engine receives a logical

operator tree as input and generates a physical operator tree as output. (Fig. 7, Col. 23, l. 40-col. 24, l. 67) (Emphasis added).

5. Andrei further discloses in Fig. 5, a table illustrating exemplary equivalence class (Eqc) level logical properties computed by the optimizer. (Col. 17, ll. 52-53).

6. Andrei also discloses ordering to apply a predicate in a plan that

> [o]ther equivalence class level logical properties are the set of all non-expansive predicates applied somewhere within the plan and the set of non-expensive predicates still to be applied by any parent . . . . if all columns involved in a given predicate are available, the predicate is applied somewhere in each sub-plan; otherwise the closest parent that has all columns available will apply the predicate.

(Col. 17, ll. 17-24).

V. ANALYSIS

From our review of the Examiner's stated rejection, we find that the Examiner appears to have set forth a detailed explanation of the rejection for a *prima facie* case of anticipation. Therefore, we look to Appellants' Briefs to show error in the proffered *prima facie* case.

Common Feature in All Claims

Independent claim 1, recites limitations, *inter alia*: "when an iterator function included in a statement is invoked . . . applying the one or more predicates to a row of data". Independent claims 14 and 27 contain similar limitations. Independent claims 6, 19, and 32 recite similar limitations: "invoking the iterator function one or more time . . . results in receiving either a row of data for which at least one predicate has been applied . . . "

Independent claims 12, 25 and 38 also recite similar limitations: "invoking the iterator function . . . obtaining a row of data that matches at least one of the one or more predicates . . . applying any of the one or more of the predicates included in the statement that have not been applied . . . " Thus, the scope of each of the independent claims all includes the following limitation: when an iterator function included in a statement is invoked (hereinafter limitation 1), then at least one predicate is applied to the row of data (hereinafter limitation 2).

The Anticipation Rejection

We now consider the Examiner's rejection of claims 1-39 under 35 U.S.C. § 102(e) as anticipated by Andrei.

We begin our analysis with claim construction. "[T]he ordinary and customary meaning of a claim term is the meaning that the term would have to a person of ordinary skill in the art in question at the time of the invention, i.e., as of the effective filing date of the patent application." *Phillips v. AWH Corp.,* 415 F.3d 1303, 1313 (Fed. Cir. 2005) (en banc). The ordinary and customary meaning of a term may be evidenced by a variety of sources, including "the words of the claims themselves, the remainder of the specification, the prosecution history, and extrinsic evidence concerning relevant scientific principles, the meaning of technical terms, and the state of the art". *Phillips v. AWH Corp.*, 415 F.3d at1314. We find no express definition of "invoke" in Appellants' Specification. Therefore, we turn to extrinsic evidence. We give the claim terminology a broad, but reasonable interpretation, thus, we construe the term "invoke" in accordance with the ordinary and customary meaning that would have been attributed to the term

by one of ordinary skill in the art at the time of the invention. Hence, we construe the term "invoke" as "[t]o activate. One usually speaks of invoking a function or routine in a program. In this sense, the term *invoke* is synonymous with call." Random House Webster's Computer & Internet Dictionary 260 (3rd ed. 1999).

Appellants contend that Andrei's disclosure of receiving a query that is passed and normalized to generate a logic operator tree, and then optimized in Fig 8A does not anticipate the limitation 1. Also Appellants contend that the cited portion, Col. 11 lines 4-13, in the Examiner's rejection describes that SQL statements are passed to the parser 360; thus, this portion does not anticipate the limitation 1. (App. Br. 8, Reply Br. 4.)

We agree with Appellants' contention. According to Andrei, executing a query statement will generally go through the stages of passing, normalizing, optimizing, and code generating. Only after code generation, will functions or routines be invoked (FF1). Further, Figs. 8A and 8B only teach the steps of the process to generate an optimized query execution plan (FF2). We find that none of the steps in the Figs 8A and 8B is related to actually invoking (activate or call) a function included in a query statement, because at this stage, the set of executable instructions is not even generated. In addition, the portion cited by the Examiner in Col. 11 lines 4-11 is only related to steps pass the query stage; thus, it is not related to invoking a function (FF1).

The Examiner further finds that:

> one could find similar meaning of "invoke" to equate to other interpretations such as when an iterator is called upon, brought forth, brought into existence, etc. Andrei explicitly states bringing about and recognizing an iterator

> in a statement (and thus the iterator is invoked). For example, Andrei discloses creating an "iterator" tree (col. 6 line 23-27) for a query execution plan. Further in this passage, Andrei discloses the query execution plan is created when a statement is received and that this plan covers the total semantics of the query (col. 6 line 36).
>
> As another example of "invoking" an iterator, Andrei explicitly discloses optimization time descriptions of [Volcano] runtime iterators (col. 24 line 30-32). Here it is seen that Volcano iterators would *have to* be invoked (i.e. brought forth from a statement) in order to provide a description of them. . . .
>
> . . . .
>
> In response, the Examiner submits that Andrei gives a clear example of the processing of a statement including an iterator function (that is invoked by being "brought forth") and predicates.

(Ans. 16-17).

Appellants further contend that merely mentioning the iterator tree by Andrei does not anticipate the limitation 1 (App. Br. 8-9, Reply Br. 4-6). In addition, Appellants also contend that merely mentioning the runtime iterators and calling into lower-level routines, such as the access method, for retrieving relevant information from the database table by Andrei does not anticipate the limitation 1 (App. Br. 9-10, Reply Br. 4-5).

We agree with Appellants' contentions. Neither arranging an iterator function in the passing, normalizing, and optimizing stage (FF1) nor creating an iterator tree that is only a data structure for a database engine to execute a query statement (FF3) constitutes invoking an iterator function included in a

query statement, because there is no activation or call of the function included in the query statement involved in those stages.

Moreover, we disagree with the Examiner's interpretation of invoking as "brought forth" (Ans. 16) a function included in a query statement, because "brought forth" a function does not activate or call the function.

Furthermore, Andrei does not explicitly nor inherently teach "Volcano iterators would have to be invoked (i.e. brought forth from a statement) in order to provide a description of them." (Ans.17). Again, "brought forth" does not activate or call, thus, as noted above, does not constitute invoke. Instead, in the stage of optimization, the search engine uses the hierarchy of physical operators and optimization time descriptions of physical operators in the physical operators model 730 for optimizing the Volcano runtime operators and eventually outputs an optimized Volcano iterator tree as the execution plan for the query statement (FF4). At this stage, the optimized Volcano iterator tree is only a data structure to be interpreted by a database engine (FF3). Hence, no Volcano runtime iterator is invoked at this stage.

Therefore, we find that the Examiner's reliance upon Andrei to reject the limitation 1 of the independent claims to be not well founded, since the Examiner has not shown, and we do not readily find where Andrei expressly or inherently teaches the limitation 1.

The Examiner also found that:

> The statement in example 1 is processed to yield a descriptive table of logical properties in figure 5. In figure 5, it is seen that the predicates included in the statement are obtained. Specifically, and for example, Equivalence class {1} includes iterator function (i.e. aggregation functions row) sum (13). Predicates from the statement (example 1, col. 13) are therefore obtained and

> described (i.e. applied or yet to be applied are descriptors of the predicates/qualification) in the table of figure 5.

(Ans. 17-18).

Appellants further contend that processing the statement of example 1 does not anticipate limitation 2, since Fig. 5 illustrates the needed ordering for each exemplary equivalence class for the sample query shown above in example 1.

We agree with Appellants' contention.

While Andrei discloses a table for ordering logical properties in Fig. 5 (FF5), the properties of the applied predicates and the predicates to be applied in the table of Fig. 5 are only to distinguish whether the predicates are applied in the same plan or will be applied by a parent plan. (FF6). Thus, Fig. 5 does not describe invoking an iterator and then applying predicates and also can not be read as applying a predicate to a row of data as claimed. In addition, the operations in the Fig. 5 table are in the optimization stage, and there is no set of executable instruction generated by code generator yet (FF1); thus, no predicates are applied to a row of data at this time. Hence, we find that the Examiner's reliance upon Andrei to reject the limitation 2 of the independent claims to be not well founded, since the Examiner has not shown, and we do not readily find where Andrei expressly or inherently teaches limitation 2.

Therefore, we agree with Appellants that the Examiner failed to show that Andrei teaches the argued feature attributed to Andrei; *i.e.*, an iterator function included in a statement is invoked and then at least one predicate is applied to the row of data, which negates anticipation. Hence, we can not sustain the anticipation rejection of independent claims 1, 6, 12, 14, 19, 25,

27, 32 and 38. The rejection of the dependent claims 2-5, 7-11, 13, 15-18, 20-24, 26, 33-37 and 39 contains the same noted deficiency. Appellants thus have demonstrated error in the Examiner's proffered prima facie case for anticipation of the subject matter of claims 1-39.

VI. CONCLUSION

For the aforementioned reasons, we conclude that Appellants have shown that the Examiner erred in determining that Andrei discloses a teaching of an iterator function included in a statement is invoked and then at least one predicate is applied to the row of data.

VII. ORDER

We reverse the anticipation rejections of claims 1-39.

REVERSED

KONRAD, RAYNES & VICTOR LLP
ATTN: IBM54
315 SOUTH BECERLY DR., STE. 210
BEVERLY HILLS, CA 90212

erc